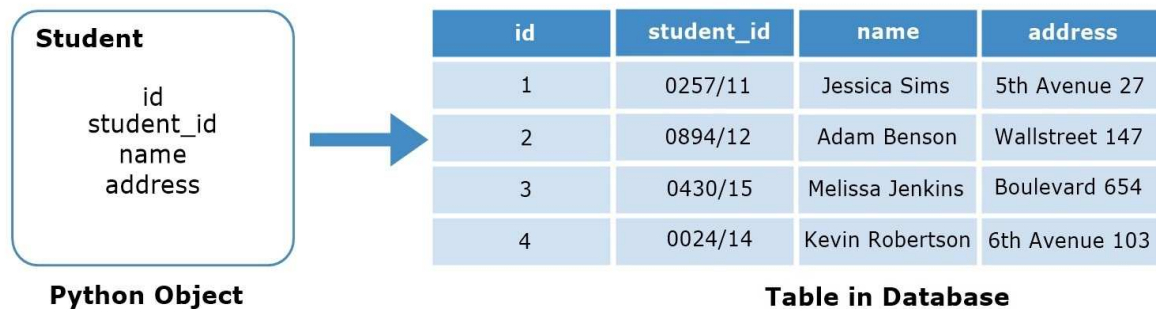


# Objektno-relaciono mapiranje

**Objektno-relaciono mapiranje (ORM)** podrazumeva biblioteku kodova koja automatizuje prenos podataka skladištenih u tabelama relacionih baza podataka u objekte koji se koriste u kodu aplikacije. ORM-ovi pružaju apstrakciju na visokom nivou na relacionoj bazi podataka koja omogućava programeru da napiše Python kod umesto SQL koda za umetanje, čitanje, ažuriranje i brisanje podataka i šema. Programeri mogu da koriste programski jezik koji im odgovara da rade sa bazom podataka umesto da pišu SQL upite ili uskladištene procedure.



Slika 10.1. Prikaz Python objekta u tabeli

Objektno-relaciono mapiranje, kao što njegovo ime nagoveštava, mapira attribute objekata u odgovarajuća polja tabele i može da ih preuzme na isti način. Prednosti ovog pristupa u odnosu na tradicionalni ogledaju se u brzom razvoju, prenosivosti baza podataka i lakoći primene. U prošlosti su web programeri za razvoj aplikacija morali da poznaju i jezike baza podataka. Poznavanje SQL-a takođe nije dovoljno, s obzirom na to da se SQL implementacije pomalo razlikuju jedna od druge u različitim bazama podataka. Ovo je postalo težak i dugotrajan proces, te su za koncept ORM-a kreirani web okviri. Dva najčešće korišćena okvira u Pythonu su Flask i Django, koji za ORM koriste **SQLAlchemy** i **Django ORM**. Na slici 10.2. prikazano je šta je potrebno za njihovu implementaciju.

WEB FRAMEWORK	NONE	FLASK	FLASK	DJANGO
ORM	SQL ALCHEMY	SQL ALCHEMY	SQL ALCHEMY	DJANGO ORM
DATABASE CONNECTOR	BUILT IN stdlib	MS SQL	psycopg	psycopg
RELATIONAL DATABASE	SQLite	MySQL	PostgreSQL	PostgreSQL

Slika 10.2. Pregled alata za rad u Flask i Django okvirima

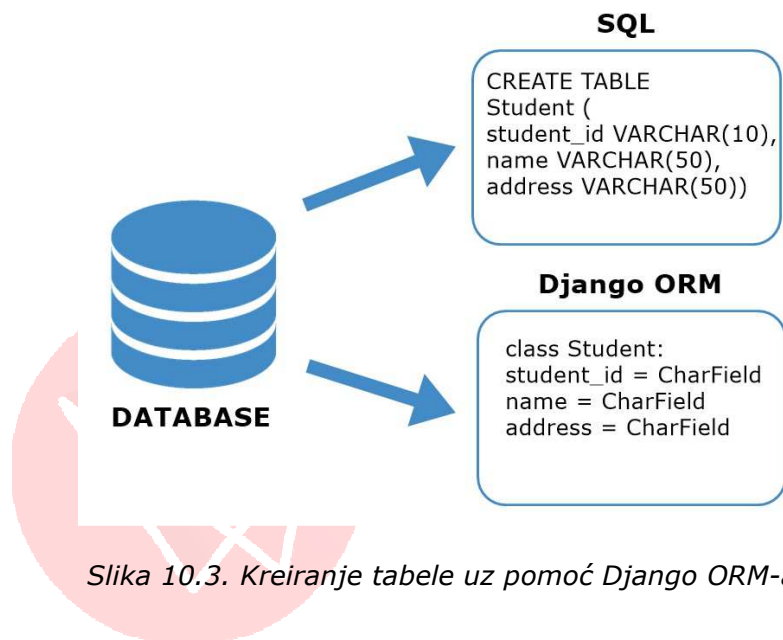
Izbor ORM-a u potpunosti zavisi od prirode i obima projekta. U većini slučajeva, programeri preferiraju SQLAlchemy. Razlog je to što je SQLAlchemy „spretniji“ za rad sa visoko složenim strukturama podataka. Django ORM je moćan na svoj način, jer pruža funkcije integrisane u Django koje su važnije od samog poboljšanja performansi.

Pošto će cela naredna lekcija biti posvećena SQLAlchemy okviru, u nastavku ćemo prikazati praktičnu primeru Django ORM-a.

## Django ORM

**Django** se isporučuje sa sopstvenim ORM-om. To je vrlo efikasan ORM koji je usko povezan sa Django okvirom. Django ORM je pogodan za obradu upita niske i srednje složenosti, iako neki smatraju da je SQLAlchemy ORM bolja opcija. Još jedna ugrađena funkcija koju Django podržava je funkcija migracija, koja je takođe deo Django ORM-a. Django je jedini okvir koji je sam po sebi potpun.

Django ORM nam daje na raspolaganje skupove upita koje možemo koristiti za preuzimanje podataka iz baze. Set upita je lista objekata modela koji služe za to da filtriraju i sređuju podatke. U nastavku je prikazano kako Django ORM zamenjuje klasično kreiranje tabele u SQL jeziku baza podataka (slika 10.3).



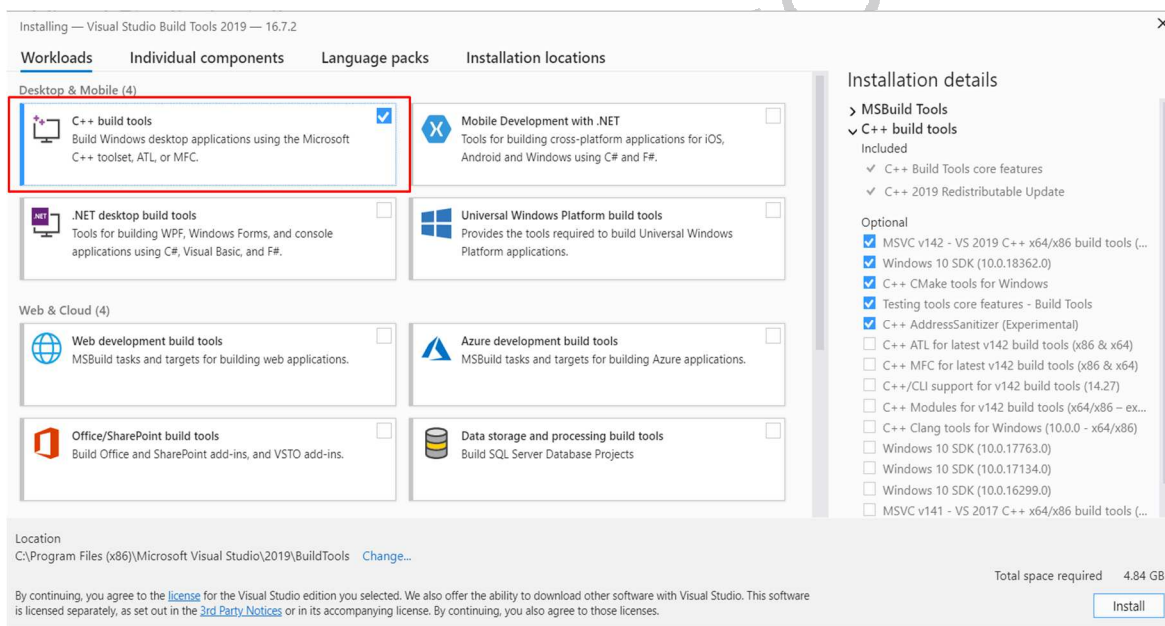
### Napomena

*Django poseduje sistem automatskog dodeljivanja identifikatora zapisa, te će se, pored navedenih atributa, odnosno kolona, u tabeli automatski dodati kolona id, koja jedinstveno identifikuje redove zapisa i predstavlja njihov primarni ključ.*

Django ORM biblioteka nije ugrađena u Python programski jezik. Stoga nam je pre korišćenja neophodna njegova instalacija. Instalaciju vršimo zadavanjem komande `pip install django` u našoj komandnoj liniji.

Korišćenje Django ORM-a prikazaćemo na primeru informacionog sistema polaganja ispita iz prethodne lekcije. Kao bazu podataka koristićemo MySQL, kao i njen server. Za razliku od dosadašnjih primera, kada smo za rad sa ovim bazama koristili `mysql.connector` adapter, u nastavku ćemo koristiti **mysqlclient**. Razlog ovome je verzija Django 3, koja je najnovija verzija koja postoji u periodu izrade ovog kursa, koja ne podržava konekcije korišćenjem biblioteke konektora. U ovom momentu se radi na poboljšanjima `mysql.connector` adaptera koji će u budućnosti biti podržani u Django 3 modulu. Instalacija `mysqlclient` modula se vrši pomoću komande `pip install mysqlclient`.

Za korišćenje `mysqlclient` modula neophodni su dodatni alati za njegovo izvršavanje. Ovaj problem je moguće rešiti instalacijom programa **Microsoft Visual C++ Build Tools**. Instalacija ovih alata dolazi zajedno sa programom Microsoft Visual Studio, koji je moguće nabaviti putem [ovog linka](#). Prilikom instalacije je neophodno označiti ove alate kao željeni dodatak uz Microsoft Visual Studio (slika 10.4).



Slika 10.4. Prikaz instalacije Microsoft Visual C++ Build Tools alata obaveznih za korišćenje `mysqlconnector` modula

### Napomena

Za korišćenje Django okvira neophodno je da Python arhitektura bude usaglašena sa arhitekturom računara. S obzirom na to da smo u prethodnim lekcijama koristili 32-bitnu Python arhitekturu, neophodno je instalirati 64-bitni Python program ukoliko vaš računar koristi tu arhitekturu.

Kada smo uspešno instalirali Django i Microsoft Visual C++ Build Tools, sledi kreiranje projekta istim postupkom kao i u lekciji *Okruženje i životni ciklus kursa Web Application Building*. U nastavku je prikazan postupak kreiranja nove aplikacije *my\_first\_app* u komandnoj liniji.

#### Postupak kreiranja aplikacije *my\_first\_app* u komandnoj liniji:

```
django-admin startproject first_project
cd first_project
python manage.py startapp my_first_app
```

U okviru kreiranog projekta podesićemo određene konfiguracije u automatski kreiranom fajlu za podešavanja tako da one odgovaraju našoj aplikaciji. Koristeći Python IDLE, otvorićemo *settings.py* kako bismo svoju aplikaciju uveli na listu aplikacija koje projekat koristi. Nova podešavanja prikazana su na slici 10.5.

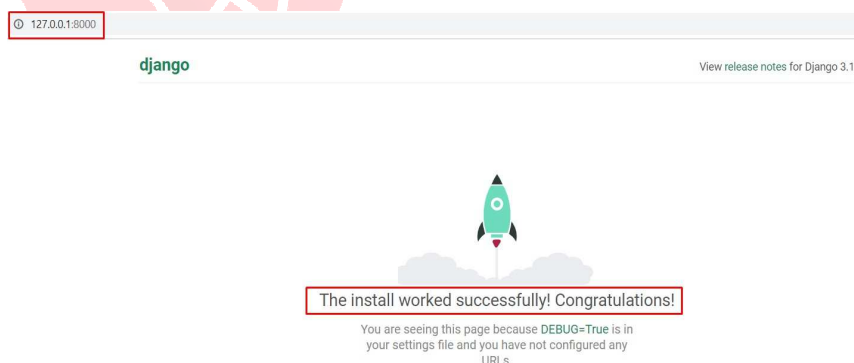
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'my_first_app'
]
```

Slika 10.5. Uvođenje nove aplikacije u *settings.py* fajlu za podešavanja

Nakon kreiranja ove aplikacije, testiraćemo uspešnost instalacije Django programa. Za testiranje instalacije neophodno je da u okviru direktorijuma projekta u komandnoj liniji pokrenemo sledeću naredbu:

```
python manage.py runserver 127.0.0.1:8000
```

Na ovaj način pokrećemo naš lokalni server na zadatoj adresi uz broj porta 8000, koji se koristi kao Django development server. Za proveru pokrenutog servera prosledićemo njegovu putanju 127.0.0.1:8000 u naš pretraživač. Ukoliko je konekcija na Django web server uspešna, na ekranu će se pojaviti slika rakete (slika 10.6).



Slika 10.6. Prikaz uspešnog povezivanja na Django web server 127.0.0.1:8000

Nakon što smo utvrdili da je naš Django program uspešno instaliran, sledi korak povezivanja na bazu podataka. Za početak ćemo kreirati bazu podataka na koju želimo da se povežemo u okviru našeg MySQL klijenta. U našem slučaju, kreiraćemo bazu podataka *example*.

Dalje je ovu bazu potrebno povezati sa projektom, što činimo u okviru podešavanja `settings.py` fajla koji se nalazi u našem projektu. U okviru objekta `DATABASE` podesićemo parametre tako da aplikacija zna da koristi `mysql` adapter i da se povezuje na bazu podataka *example* na port 8000 lokalnog servera. Podešavanja bi sada trebalo da izgledaju kao na slici 10.7.

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'example',  
        'USER': 'root',  
        'PASSWORD': '',  
        'HOST': '127.0.0.1',  
        'PORT': '3306',  
    }  
}
```

*Slika 10.7. Podešavanje parametara baze podataka u okviru `settings.py` fajla za podešavanje projekta*

Sa ovako podešenim okruženjem možemo da počnemo sa definisanjem klasa. Deklaracija klasa u Django se vrši u okviru `models.py` fajla aplikacije. Za korišćenje modela potrebno je uvođenje modula **models** u program. Ovaj model omogućava definisanje tabela, atributa i relacija pozivanjem odgovarajućih ugrađenih klasa.

Klasa `Model` predstavlja jedinstveni, konačni izvor informacija o podacima. Generalno se svaki objekat `Model` preslikava u jednu tabelu baze podataka. Za preslikavanje tabele na osnovu deklarisanе klase neophodno je prosleđivanje modela u vidu parametra.

Deklaracija atributa, koji zapravo predstavljaju kolone tabela, koristi ugrađene klase tipova podataka koje za parametar primaju ograničenja. Listu svih podržanih tipova možete videti na [ovom linku](#).

Povezivanje tabela takođe se vrši pozivanjem ugrađenih klasa modula `models` koje su tome namenjene. Ove klase kao obavezan parametar uzimaju objekat tabele za koju se vezuju, dok su ostali parametri opcioni. Parametar `on_delete` u našem slučaju nalaže da, ukoliko instanca studenta na koju automobil ima referencu bude obrisana, treba da bude obrisana i on. U suprotnom, automobil ne bi imao objekat na koji će da se referencira i njegovo postojanje ne bi bilo smisleno. Parametar `related_name` predstavlja naziv kolone atributa koji će biti referenca na objekte studenata. Sve moguće klase relacija u Django možete videti na [ovom linku](#).

U nastavku je prikazan primer deklaracije klasa studenata i vozila.

### Primer deklaracije klasa u okviru module.py fajla aplikacije:

```
from django.db import models

class Student(models.Model):
    student_id = models.CharField(max_length=10)
    name = models.CharField(max_length=50)
    address = models.CharField(max_length=50)

class Car(models.Model):
    brend = models.CharField(max_length=45)
    student = models.OneToOneField(
        Student,
        on_delete=models.CASCADE,
        related_name='car'
    )
```

Ovako kreirane modele neophodno je registrovati u admin.py fajlu aplikacije. U nastavku su prikazani njihovo uvođenje u fajl i njihova registracija.

### Primer registracije klasa u okviru admin.py fajla aplikacije:

```
from django.contrib import admin
from .models import Student, Car

admin.site.register(Car)
admin.site.register(Student)
```

Naše klase su sada spremne za upisivanje u bazu podataka. Za upisivanje podataka u bazu koristimo funkciju migracije. **Migracije** u Djangu zadužene su za apliciranje promena nad modelima. Za početak ćemo na bazu aplicirati manage.py fajl, zamenski fajl Django admina koji ujedno postavlja i DJANGO\_SETTINGS\_MODULE koji ima referencu na settings.py fajl projekta. U ovu svrhu koristi se komanda `python manage.py migrate`. Rezultati ove migracije prikazani su na slici 10.8.

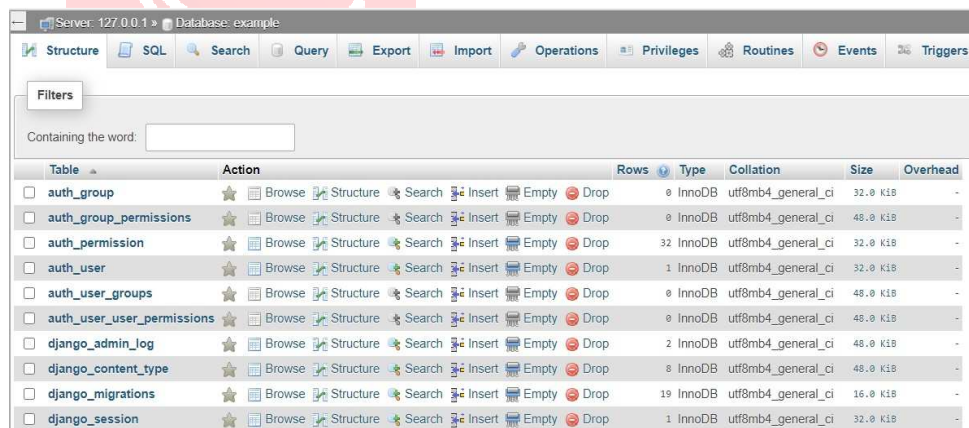


Table	Action	Rows	Type	Collation	Size	Overhead
auth_group	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
auth_group_permissions	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
auth_permission	Browse Structure Search Insert Empty Drop	32	InnoDB	utf8mb4_general_ci	32.0 KiB	-
auth_user	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 KiB	-
auth_user_groups	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
auth_user_user_permissions	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
django_admin_log	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	48.0 KiB	-
django_content_type	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	48.0 KiB	-
django_migrations	Browse Structure Search Insert Empty Drop	19	InnoDB	utf8mb4_general_ci	16.0 KiB	-
django_session	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 KiB	-

Slika 10.8. Prikaz rezultata migracije manage.py fajla projekta u example bazu podataka u phpMyAdmin klijentu



Poslednji korak u ovom procesu je apliciranje promena nad bazom, odnosno kreiranje tabela koje smo definisali u aplikaciji. Apliciranje promena izvršenih u aplikaciji vršimo pomoću komande `python manage.py makemigrations my_first_app`. Nakon ovoga potrebno je ponovo izvršiti `python manage.py migrate` kako bi se ove promene aplicirale na bazu. Konačni rezultat celog procesa prikazan je na slici 10.9.

Server: 127.0.0.1 » Database: example

Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers

Filters

Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> auth_group	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 K1B	-
<input type="checkbox"/> auth_group_permissions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 K1B	-
<input type="checkbox"/> auth_permission	★ Browse Structure Search Insert Empty Drop	32	InnoDB	utf8mb4_general_ci	32.0 K1B	-
<input type="checkbox"/> auth_user	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 K1B	-
<input type="checkbox"/> auth_user_groups	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 K1B	-
<input type="checkbox"/> auth_user_user_permissions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 K1B	-
<input type="checkbox"/> django_admin_log	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	48.0 K1B	-
<input type="checkbox"/> django_content_type	★ Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	48.0 K1B	-
<input type="checkbox"/> django_migrations	★ Browse Structure Search Insert Empty Drop	19	InnoDB	utf8mb4_general_ci	16.0 K1B	-
<input type="checkbox"/> django_session	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 K1B	-
<input type="checkbox"/> my_first_app_car	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 K1B	-
<input type="checkbox"/> my_first_app_student	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 K1B	-
12 tables	Sum	65	InnoDB	utf8mb4_general_ci	432.0 K1B	0 B

Slika 10.9. Prikaz rezultata apliciranja promena izvršenih u aplikaciji nad example bazom podataka u phpMyAdmin klijentu

## Pitanje

U kom od navedenih parametara DATABASES podešavanja u okviru settings.py fajla projekta podešavamo adapter?

- NAME ;
- HOST ;
- **ENGINE ;**
- PORT ;

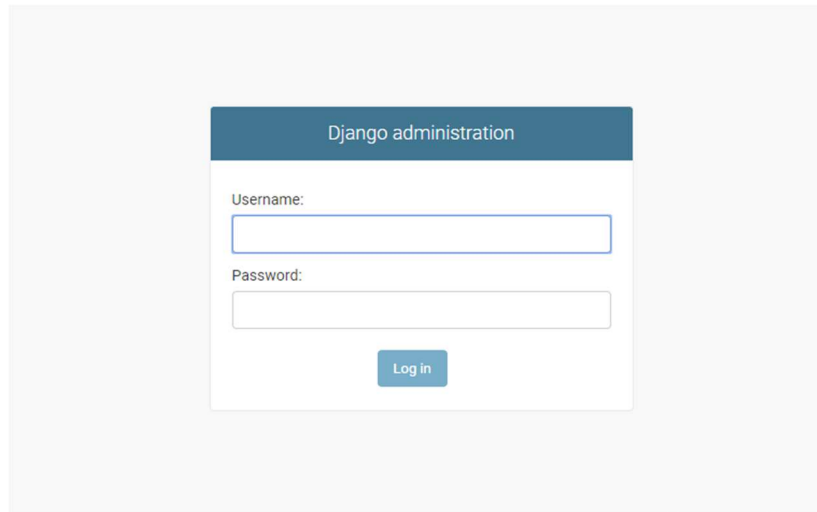
## Objašnjenje:

Adapter baze podataka u DATABASES podešavanjima settings.py fajla projekta podešavamo u parametru **ENGINE**.

Za prikaz baze podataka, Django, pored klijenta baze podataka, koristi i sopstveni *admin panel* na svom web serveru, kojem možemo pristupiti pomoću sledeće putanje:

**localhost:8000/admin**

Otvaranjem ove putanje dospevamo na log in stranicu Django admina (slika 10.10).



*Slika 10.10. Log in stranica Django administratora*

Za pristup ovoj stranici neophodno je kreiranje superusera, odnosno korisnika koji će imati sve privilegije u vezi sa manipulacijom podacima iz baze. Kreiranje ovog korisnika vrši se pomoću komande `python manage.py createsuperuser` u okviru direktorijuma projekta. Ovako zadata komanda zatražiće od nas informacije o željenom korisničkom imenu, e-mail adresi i lozinci. Jednom unete informacije kreiraće našeg superusera u bazi korisnika. Ovako unete informacije koristimo za pristup panelu (slika 10.11).



*Slika 10.11. Početna strana Django administrativnog panela*

Pomoću dugmeta + pored imena tabela dodaćemo jednog studenta, a potom i automobil koji će za atribut `Student` uzeti objekat tog studenta, odnosno referencirati na njega.



Home > My\_First\_App > Students > Add student

**ADD student**

Student id:

Name:

Address:

---

Home > My\_First\_App > Cars > Add car

**ADD car**

Brend:

Student:

Slika 10.12. Dodavanje objekata u tabelu korišćenjem Django administrativnog panela

Izmene u ovom panelu biće automatski ažurirane i u phpMyAdmin klijentu, što možemo videti na slici 10.13.

+ Options

	id	student_id	name	address
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2	0335/12	Sam Garcia	1st Avenue 2234

☐ Check all
 With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

☐ Show all | Number of rows: 25 Filter rows: Search this table

---

+ Options

	id	brend	student_id
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2	Porsche	2

☐ Check all
 With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

☐ Show all | Number of rows: 25 Filter rows: Search this table

Slika 10.13. Prikaz automatskog ažuriranja podataka u phpMyAdmin klijentu baze podataka

Kao što možemo primetiti na slici 10.13, automobil ima referencu na studenta koju skladišti u koloni *student\_id*. Drugim rečima, ovaj automobil vezan je za tačno jednog studenta i to je u našem slučaju *Sam Garcia*, čiji primarni ključ, odnosno id, ima vrednost 2. Kardinalnost koja je definisana u našoj tabeli glasi da ako student ne postoji, ne može da postoji ni auto. Iz ovog razloga, brisanje studenta nije dozvoljeno dokle god postoji automobil koji ima referencu na njega. U nastavku je prikazana greška izazvaka brisanjem studenta na koga referencira automobil (slika 10.14).



Slika 10.14. Greška izazvana pokušajem brisanja studenta na koga automobil ima referencu

## Rezime

- Objektno-relaciono mapiranje (ORM) podrazumeva biblioteku kodova koja automatizuje prenos podataka skladištenih u tabelama relacionih baza podataka u objekte koji se koriste u kodu aplikacije.
- Dva najčešće korišćena okvira u Pythonu su Flask i Django, koji za ORM koriste SQLAlchemy i Django ORM.
- SQLAlchemy je „spretniji“ za rad sa visoko složenim strukturama podataka;
- Django je pogodan za obradu upita niske i srednje složenosti. Podržava funkciju migracija koja je deo Django ORM-a. Jedini je okvir koji je sam po sebi potpun.
- Upotreba Django okvira u MySQL bazama podataka zahteva instalaciju:
  - Django softvera;
  - MySQL Servera;
  - mysqlclient adaptera;
  - Microsoft Visual C++ Build Tools alata.
- Kreiranje projekta u Django okviru vrši se komandom `django-admin startproject project_name`.
- Kreiranje aplikacije u Django okviru vrši se komandom `python manage.py startapp app_name`.
- Za korišćenje aplikacije obavezno je njeno dodavanje na spisak aplikacija u okviru `settings.py` fajla projekta.
- Provera instalacije obavlja se komandom: `python manage.py runserver 127.0.0.1:8000`.
- U okviru `settings.py` fajla treba podesiti parametre baze podataka: adapter, ime baze, server i port.
- Definisanje klasa vrši se u okviru fajla `models.py`. Modul `models` sadrži sledeće osnovne funkcije:

- definisanje tabele – `modules.Module;`
  - definisanje kolone – `modules.CharField(constraint);`
  - definisanje veze – `modules.ManyToMany(object).`
- Migracije apliciraju promene u bazi podataka:
    - `python manage.py migrate` – apliciranje `manage.py` fajla, zamenskog fajla Django admina;
    - `python manage.py makemigrations app_name` – apliciranje promena nad imenovanom bazom podataka.
  - *Django admin panel* je Djangov interfejs koji omogućava manipulaciju podacima u bazi na Django serveru. Nalazi se na adresi *localhost:8000/admin*.

